มาตรฐานการสื่อสารข้อมูลและการพัฒนายูทิลิตี้สำหรับไมโครคอมพิวเตอร์

ดร.ไพศาล สงวนหมู่

พื้นฐานเกี่ยวกับการรับส่งข้อมูลแบบอนุกรม

การสื่อสารเป็นหนทางที่จะแลกเปลี่ยนข่าวสาร และรับข้อมูลเข้าสู่ระบบ ไม่ว่าจะเป็นชีวิตประจำวันของมนุษย์ หรือในโลกของคอมพิวเตอร์หากขาดการติดต่อสื่อสารเสียแล้ว ก็เหมือนกับปิดตัวเองไว้ในห้องขังไม่ได้รับรู้กับโลกภายนอก คอมพิวเตอร์เองก็เหมือนกัน จะต้องมีการสื่อสารกับโลกภายนอกเพื่อติดต่อรับคำสั่ง หรือส่งผลการทำงานออกมาให้เห็นไม่เช่นนั้น คอมพิวเตอร์ก็ไม่มีประโยชน์อะไร

การสื่อสารในโลกของคอมพิวเตอร์เป็นไปได้หลายแบบ(เช่นเดียวกับมนุษย์ก็มีการสื่อสารหลายแบบ) ยกตัวอย่างที่เห็นได้ง่าย ๆ ในไมโครคอมพิวเตอร์ซีพียูจะต้องติดต่อสื่อสารกับแป้นพิมพ์เพื่อรับข้อมูลและคำสั่ง ต้องติดต่อสื่อสารกับจอภาพเพื่อส่งผลที่ต้องการแสดงออกบนจอ ต้องติดต่อสื่อสารกับดิสก์เพื่ออ่านและเขียนข้อมูลลงบนดิสก์และอื่น ๆ อีกมาก นั่นเป็นการติดต่อสื่อสารภายในระบบเอง เพื่อให้ระบบทำงานได้ นอกจากการติดต่อสื่อสารภายในระบบแล้ว การติดต่อสื่อสารกับโลกภายนอก(ของสังคมคอมพิวเตอร์) ก็มีความจำเป็นเพื่อเพิ่มประสิทธิภาพของเครื่องให้สูงขึ้น เนื่องจากคอมพิวเตอร์รับรู้ข้อมูลในแบบของดิจิตอลจึงติดต่อกันแบบดิจิตอล การติดต่อกันระหว่างคอมพิวเตอร์จึงน่าจะเรียกว่าการสื่อสารข้อมูล เพราะคอมพิวเตอร์แลกเปลี่ยนข้อมูลซึ่งกันและกันโดยตัวคอมพิวเตอร์เองยังไม่มีความรู้สึกยินดียินร้ายกับข้อมูลที่ได้รับ

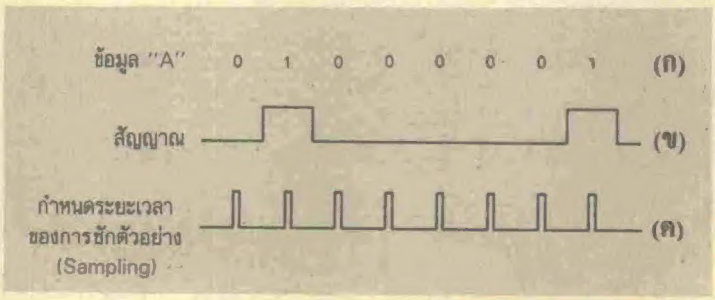
การสื่อสารแบบอนุกรม

คอมพิวเตอร์ทำงานโดยการวิ่งของอิเล็กทรอนิกส์พัลส์ ดังนั้นการส่งข้อมูลออกจากคอมพิวเตอร์หรือรับเข้าสู่คอมพิวเตอร์จะต้องอยู่ในรูปของอิเล็กทรอนิกส์พัลส์ การที่จะส่งข้อมูลติดต่อกันแบบขนาน คือส่งไปพร้อมกัน 8 บิต จะต้องมีวงจรไฟฟ้าต่อถึงกัน 8 วงจร คืออย่างน้อยต้องมีสายการติดต่อถึง 9 เส้น (ไว้เป็นสัญญาณอ้างอิง 1 เส้น) ถ้าหากการส่งไปเป็นระยะทางไกล ๆ ก็จะทำให้เกิดความสิ้นเปลืองในการเชื่อมต่อวงจรทั้ง 8 อีกทั้งเครือข่ายการสื่อสารที่มีอยู่ (ระบบโทรศัพท์) ก็เป็นแค่ระบบวงจรเดียว ทำให้การสื่อสารแบบขนานไม่เหมาะสำหรับการสื่อสารระยะไกล

แทนที่จะใช้วงจรทางไฟฟ้า 8 วงจร ถ้าเราทยอยส่งออกไปทีละพัลส์หรือทีละบิต เราก็จะใช้แค่วงจรเดียว ทำให้ลดความสิ้นเปลืองในการเชื่อมต่อวงจรลง การสื่อสารแบบนี้จึงเรียกว่าแบบอนุกรม เพราะทยอยส่งทีละบิต

สมมุติว่าเราแทนข้อมูลที่เป็น 1 ด้วยสัญญาณไฟฟ้าที่สูง 5 V และข้อมูล 0 ด้วยสัญญาณไฟฟ้าอักษร A ซึ่งมีรหัส ASCII เป็น 01000001 ก็จะมีรูปร่างดังในรูปที่ 1(ก)

ปัญหาทางฝ่ายรับก็คือว่า ฝ่ายรับจะรู้ได้อย่างไรว่า 0 หรือ 1 ติดต่อกันกี่ตัว เพราะไม่มีการเปลี่ยนแปลงของสัญญาณช่วงนั้น ปัญหานี้แก้ได้โดยการกำหนดระยะห่างระหว่างพัลส์ที่จะส่งให้เป็นที่ตกลงกันทั้งฝ่ายรับและฝ่ายส่ง นั้นก็คือกำหนดความเร็วในการรับส่งข้อมูลนั้นเอง เพื่อที่ฝ่ายรับจะได้กำหนด



รูปที่ 1 ลักษณะการส่งข้อมูลในรูปสัญญาณไฟฟ้า

ชักตัวอย่างหลายเท่าของระยะเวลาที่ pling) สัญญาณที่เข้ามาได้อย่างถูกต้อง ในรูปที่ 1(ค) ในวงจรการรับบางทีเขา ระยะเวลาของการชักตัวอย่าง (Sam- กำหนดให้ เพื่อความถูกต้องของข้อมูล ที่ได้รับ เช่นเป็น 2,4,8,16,32 หรือ 64 เท่า

ความเร็วของการส่งสัญญาณจึงมีหน่วยเป็น Bit Per Second หรือ BPS บางครั้งท่านอาจจะสับสนกับคำว่า Baud Rate ความจริงแล้ว Baud คืออัตราการเปลี่ยนของสัญญาณใน 1 วินาที ซึ่งในทางดิจิตอลแล้วสัญญาณมีได้แค่สองระดับ การเปลี่ยนแปลงของสัญญาณ 1 ครั้ง ให้ข้อมูลได้ 1 บิต ดังนั้น Baud Rate จึงมีค่าเท่ากับ Bit Rate แต่ถ้าเราเข้ารหัสสัญญาณดิจิตอลโดยแทนด้วยสัญญาณทางไฟฟ้า เช่น ถ้าเราให้สัญญาณทางไฟฟ้าที่มีความสูง 4 ระดับ ใน 1 ระดับของสัญญาณไฟฟ้าจึงแทนได้ด้วยเลขฐานสอง 2 บิต นั่นคือ การเปลี่ยนแปลงของสัญญาณทางไฟฟ้า 1 ครั้งจะให้ข้อมูล 2 บิต ในกรณีเช่นนี้ Bit Rate จะมีค่าเป็น 2 เท่าของ Baud Rate ลองหารายละเอียดเกี่ยวกับโมเด็มมาศึกษาดูจะพบว่าโมเด็มบางค่าส่งข้อมูลด้วยความเร็ว 1200 baud แต่อัตราการส่งข้อมูลมีถึง 9600 BPS ในช่วงนี้เอาเป็นว่าความเร็วในการรับส่งข้อมูลถือว่ามีหน่วยเป็น baud นั่นคือ 1 วินาที ส่งไปได้กี่พัลส์

หมดปัญหาเรื่องการกำหนดขอบเขตของข้อมูล จากสัญญาณที่รับมาได้ของฝ่ายรับแล้ว คราวนี้ก็มาถึงปัญหาที่ว่าบิตไหนเป็นบิตต้นบิตไหนเป็นบิตท้ายของข้อมูลละ! เพราะข้อมูลมันจะต้องอยู่กันเป็นชุด อย่างน้อย 7-8 บิต จึงจะใช้แทนอักขระหรือข้อมูลได้ ปัญหานี้ก็คือปัญหาการกำหนดขอบเขตของอักขระ (1 หน่วยของข้อมูล) ว่าอยู่ในช่วงไหนแน่ของข้อมูลที่เป็นสายธารของ 0 และ 1 มีวิธีการที่กำหนดขอบเขตของการแยกอักษรอยู่สองแบบที่ใช้กันอยู่ในขณะนี้ ซึ่งกำหนดเป็นวิธีการในการส่งข้อมูลแบบอนุกรม วิธีการที่ว่านั้นก็คือ การส่งแบบอสมวาร (Asynchronous) และสมวาร (Synchronous)

การส่งแบบ อสมวาร (Asynchronous) นั้น แต่ละ "อักขระ" ที่ส่งออกไปนั้นไม่จำเป็นต้องต่อเนื่อง และช่วงเวลาระหว่างอักขระไม่กำหนดแน่นอนแต่สัญญาณที่ส่งออกไปนั้นมีตัวบอก จุดเริ่มต้นและจุดสุดท้ายของอักขระบวกเข้าไปด้วย ซึ่งเรียกว่า บิตเริ่ม (Start bit) และบิตหยุด (Stopbit) เพื่อบอกฝ่ายรับทราบถึงขอบเขตของการแยกอักขระได้ ตามข้อตกลงมาตรฐานของสมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (EIA) กำหนดมาตรฐาน RS 232C เกี่ยวกับสัญญาณที่ใช้ในการส่งไว้ให้เป็นที่เข้าใจและนำไปใช้เข้ากันได้ดังนี้

- ข้อมูลที่เป็น 1 ให้แทนด้วย -15 V
- ข้อมูลที่เป็น 0 ให้แทนด้วย +15V
- บิตเริ่ม (Start bit) เริ่มต้นเมื่อสัญญาณเปลี่ยนจากสูง (+15V) ไปต่ำ (-15V)
- บิตหยุด (Stop bit) บอกว่าสิ้นสุดขอบเขตของข้อมูลแล้ว โดยจะมีสัญญาณแบบ High เป็นระยะเวลา 1 ช่วงนาฬิกา (clock) หรือ $1\frac{1}{2}$ ช่วงนาฬิกา

ลักษณะของข้อมูลที่ส่งแบบอนุกรมจึงมีหน้าตาดังรูปที่ 2

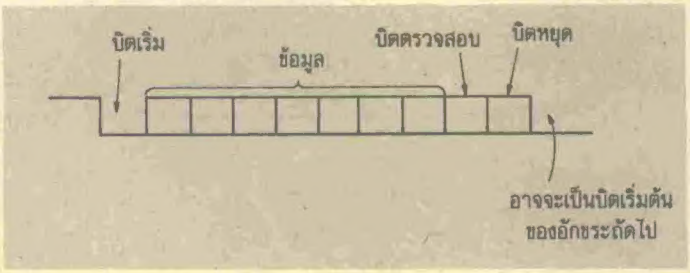
ดังนั้นการกำหนดพารามิเตอร์สำหรับการสื่อสารอนุกรมแบบอสมวารจึงต้องประกอบด้วย

1. ความเร็วของการส่ง (Baud)
2. ขนาดของข้อมูลที่รวมเป็น 1 อักขระ (Data)
3. ชนิดของการตรวจสอบ (Parity)
4. ความยาวของบิตหยุด (Stop bit)

สำหรับการรับส่งแบบอนุกรมชนิดสมวาร หรือ Synchronous นั้น แตกต่างจากอสมวารตรงที่ไม่มีทั้งบิตเริ่มและบิตหยุด อักขระทุกอักขระที่ส่งจะต้องส่งกันมาอย่างต่อเนื่องจนครบจำนวนที่ตกลงกันเอาไว้ จะหยุดพักชั่วขณะกลางคันไม่ได้ การกำหนดขอบเขตของอักขระ ขึ้นอยู่กับข้อตกลงเกี่ยวกับการรับส่งข้อมูลที่เรียกกันว่า โปรโตคอล (Protocol) หลายท่านคงเคยได้ยินได้ฟังคำคำนี้มาแล้วไม่มากก็น้อย อาจจะสงสัยว่า อะไรกันแน่ที่เรียกว่าโปรโตคอล ผมขอหยุดพักเรื่องการส่งข้อมูลแบบสมวารก่อน ลงมาคุยให้ฟังเรื่องโปรโตคอลนิดหนึ่ง

โปรโตคอล คือข้อตกลงหรือกฎเกณฑ์ที่จะต้องปฏิบัติระหว่างฝ่ายรับและฝ่ายส่งในเรื่องต่อไปนี้

1. ขอบเขตของอักขระ
2. การตรวจสอบข้อผิดพลาดและการตอบสนองต่อความผิดพลาด
3. ลำดับของข้อมูลที่ส่ง
4. การส่งข้อมูลที่เหมือนกับอักขระควบคุม



รูปที่ 2 รูปแบบการส่งข้อมูลแบบอสมวาร

5. การควบคุมการรับส่งในสายใครจะส่งก่อนหลังใคร

6. กรณีพิเศษ เช่น กรณีไม่มีข้อมูลจะส่งควรจะทำอะไร

7. เวลาที่ต้องรอการตอบสนองจะต้องทดลองส่งใหม่กี่หนกรณีที่ไม่มีการตอบสนอง

8. การเริ่มการติดต่อ

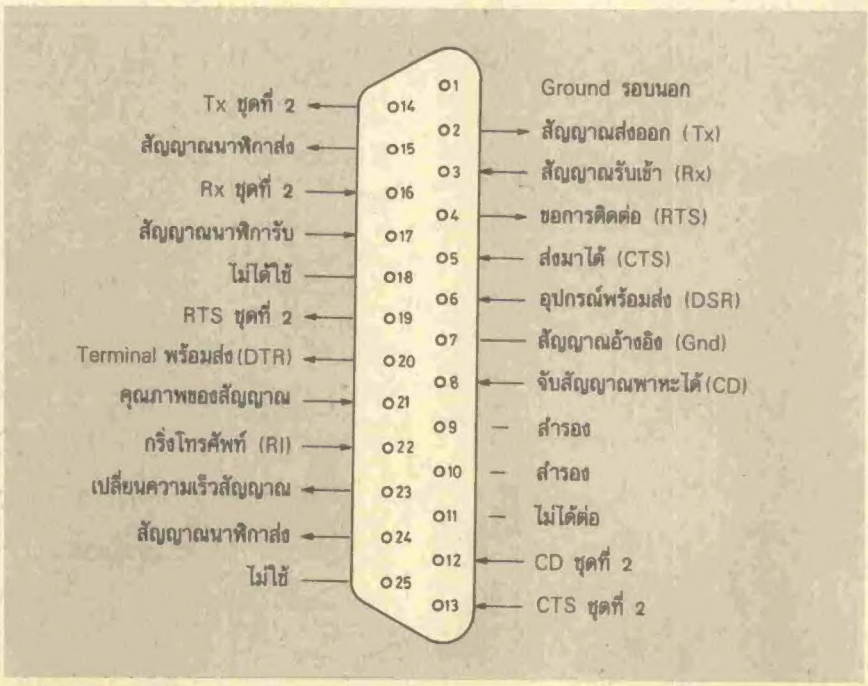
มีการกำหนดโปรโตคอลขึ้นมาใช้กับการสื่อสารแบบอนุกรมหลายแบบ เช่น Bisync, HDLC, SDLC เป็นต้น ลักษณะที่สำคัญของโปรโตคอลเหล่านี้ก็คือ จะต้องมีอักขระพิเศษนำหน้าก่อนการส่งซึ่งเรียกว่า Sync ซึ่งมีรูปแบบของพัลส์ 0 และ 1 รูปแบบหนึ่งที่แน่นอน เพื่อให้ฝ่ายรับได้รู้ว่าขอบเขตของอักขระมีดังนี้ ฝ่ายรับจะต้องพยายามหารูปแบบของข้อมูลที่เป็น Sync ให้ได้ข้อมูลที่ตามมาก็แยกเอาเองทีละ 8 ก็แล้วกัน (ฝ่ายส่งจึงหยุดชั่วขณะไม่ได้) จนกว่าจะครบจำนวนที่ตกลงกันเองไว้ หรือจนกว่าจะมีอักขระพิเศษบอกว่าหมดข้อมูลแล้ว

ถ้าหากข้อมูลมีหน้าตาเหมือนกับ Sync หรืออักขระที่ใช้ในการควบคุมขึ้นมาละทำยังไง ยุ่งละซิ! โปรโตคอลแต่ละชนิดมีวิธีแก้ปัญหานี้ไม่เหมือนกัน ถ้าเป็น Bisync ให้เพิ่มอักขระพิเศษที่เรียกว่า DLE (Data Link Escape) เข้าไป เพื่อบอกให้ฝ่ายรับรู้ว่า อักขระที่ตามมาเป็นข้อมูลนะไม่ใช่อักขระที่ใช้ในการควบคุม สำหรับ HDLC กำหนด จำนวนข้อมูลที่แน่นอนไปเลยว่าจะส่งกี่อักขระ ฝ่ายรับจะได้เตรียมตัวได้ถูกต้องว่าช่วงไหนเป็นข้อมูลจริง ๆ

สำหรับการส่งข้อมูลแบบสมวาร (Synchronous) จะไม่ขอกล่าวเพิ่มเติมในที่นี้เพราะส่วนมากจะเป็นการสื่อสารสำหรับเครื่องระดับมินิจนถึงเมนเฟรมเท่านั้น สำหรับเครื่องระดับไมโครนั้นการติดต่อสื่อสารส่วนมากจะเป็นแบบอสมวารเสียมากกว่า นอกจากจะใช้ไมโครไปติดต่อกับเครื่องระดับเมนเฟรม

มาตรฐาน RS 232C

เนื่องจากมีหลายบริษัทผลิตอุปกรณ์การสื่อสารออกมาสู่ท้องตลาด เพื่อให้อุปกรณ์ดังกล่าวใช้ร่วมหรือใช้แทนกันได้ สมาคมผู้ผลิตอุปกรณ์อิเล็กทรอนิกส์ (Electronic Industrial Association) จึงได้กำหนดมาตรฐานที่ 232 และเรียกว่า Recommended Standard 232 หรือ RS 232 ขึ้นมา ซึ่งกำหนดถึง คุณลักษณ์ทางไฟฟ้า และข้อต่อของอุปกรณ์สำหรับการสื่อสารแบบอนุกรมขึ้นมา ซึ่งได้กล่าวไปบ้างแล้วในหัวข้อก่อนนี้ สำหรับคุณลักษณะทางไฟฟ้าจะขอข้ามไป เพราะเป็นหน้าที่ของวิศวกรเท่านั้นที่ควรจะรู้ สิ่งที่เราควรจะรู้ก็คือ สัญญาณที่ออกมาจากข้อต่อของอุปกรณ์ชนิดนี้ เขาแนะนำให้ใช้สำหรับทำอะไร และต่อไว้ที่ขาไหนของข้อต่อบ้าง



รูปที่ 3 การกำหนดมาตรฐานของข้อต่อ RS 232

สำหรับการติดต่อแบบอนุกรม ที่ใช้มาตรฐาน RS 232C เขาจะให้ชื่ออุปกรณ์ที่จะใช้ในการรับส่งว่า ดาต้าเทอร์มินัล ในที่นี้ก็คือตัวไมโครคอมพิวเตอร์ และอุปกรณ์ที่ทำหน้าที่ส่งสัญญาณไปตามสายว่าเป็นดาต้าเซต ถ้าหากเป็นการส่งไปตามสายโทรศัพท์ดาต้าเซต ก็คือ โมเด็ม

พิธีในการติดต่อเขาก็กำหนดไว้เป็นข้อแนะนำว่าควรจะทำดังนี้

เมื่อเริ่มต้น

ดาต้าเทอร์มินัล ส่งสัญญาณ DTR หรือ Data Terminal Ready (ขา 20) ออกไปบอกดาต้าเซต

ดาต้าเซตถ้าพร้อมที่จะทำงานจะมีสัญญาณพร้อมที่ DSR หรือ Data Set Ready (ขา 6)

เมื่อจะส่งข้อมูลใด ๆ

ดาต้าเทอร์มินัลส่งสัญญาณการติดต่อ RTS (Request to Send) (ขา 4) ไปบอกดาต้าเซต

ดาต้าเซตถ้าพร้อมจะส่งสัญญาณ CTS (clear to Send) (ขา 5) มาบอกดาต้าเทอร์มินัล

ดาต้าเทอร์มินัลเริ่มส่งข้อมูลออกที่ ขา 2

ในขณะใดๆ ก็ตามที่ Data รับสัญญาณได้ จะส่งสัญญาณไปบอกดาต้าเทอร์มินัลดังนี้

RI Ring Indicator เมื่อมีกริ๊งสัญญาณมาตามสาย

CD Cerrier Detected เมื่อรับ Tone จากฝ่ายตรงข้ามได้

RX ข้อมูลที่ดาต้าเซตแปลงเป็นดิจิตอลแล้ว

สัญญาณที่ใช้ในการกำหนดให้ทำงานพร้อมกันสำหรับดาต้าเซตและดาต้าเทอร์มินัล โดยดาต้าเทอร์มินัลเป็นผู้กำหนด มีดังนี้

- สัญญาณนาฬิกาส่ง
- สัญญาณนาฬิการับ
- สัญญาณขอเปลี่ยนความเร็ว

อะแดปเตอร์สื่อสารแบบอนุกรมของไอบีเอ็มพีซี

สำหรับอะแดปเตอร์สื่อสารของเครื่องไอบีเอ็มพีซี มีลักษณะเป็นแผงวงจรเสียบลงในสล้อตที่ว่างของเครื่อง มีหลายบริษัทผลิตขึ้นมา ที่เราจะพบในเมืองไทยก็ส่วนมากจะมาจากไต้หวันหรือญี่ปุ่น บางทีช่องการสื่อสารแบบอนุกรมก็ติดมากับแผงวงจรอเนกประสงค์ที่มีหลายฟังก์ชันในตัว สำหรับอะแดปเตอร์สื่อสารของไอบีเอ็มเองนั้นต่างจากมาตรฐาน RS 232 ตรงที่ระดับของสัญญาณที่ส่งออกแทนที่จะเป็น +15V และ -15V กลับเป็นแค่ 5 V และ 0 V และเพิ่มการรับส่งแบบกระแสวนกลับ 20 MA ไว้ให้ผู้ใช้ได้เลือกลักษณะของการติดต่อด้วย ข้อดีของการส่งแบบกระแสวนกลับก็ตรงที่ระยะทางของการส่งจะไปได้ไกลกว่าแรงดันไฟฟ้าที่ใช้ใน RS 232 อีกทั้งใช้การเชื่อมต่อส่งข้อมูลโดยใช้ลำแสงแบบ opto coupling ทำให้แยกวงจรรับและส่งออกจากกันได้โดยเด็ดขาด อย่างไรก็ตามการรับส่งสัญญาณแบบกระแสวนกลับไม่ค่อยมีให้ใช้กันในอุปกรณ์ทั่ว ๆ ไป เท่าที่ผมเคยพบก็มีแค่อุปกรณ์ทางด้านห้องทดลองบางชิ้นที่ออกแบบมานานแล้ว เหมาะสำหรับการแยกวงจรรับและส่งออกจากกัน เช่นในกรณีที่ต้องรับสัญญาณที่ผ่านสิ่งแวดล้อมที่อาจจะทำให้เกิดไฟแรงสูงหลงเข้ามาในสายสื่อสารได้เป็นต้น

หัวใจสำคัญของอะแดปเตอร์การสื่อสารก็คือไอซี 8250 ซึ่งมีทั้งตัวกำหนด Baud Rate ที่สามารถควบคุมได้ในตัว และมีรีจิสเตอร์ที่ใช้ในงานควบคุมไว้ 7 รีจิสเตอร์ด้วยกัน ซึ่งจะได้กล่าวในรายละเอียดต่อไป นอกเหนือจากนั้นก็เป็นพวกไอซีซึ่งทำหน้าที่แยกตำแหน่งที่อยู่ของช่องทางออก และ crystal oscillator 1.8432 MHz ไอบีเอ็มพีซีสำรองช่องการสื่อสารแบบอนุกรมไว้ 2 พอร์ต โดยดอสจะรับรู้ว่ามีช่องการสื่อสารแบบอนุกรมเพียง 2 ช่องทางเท่านั้น (รวมทั้ง ภาษาเบสิกด้วย) แต่ก็ไม่ได้หมายความว่า เราจะทำให้มันสื่อสารแบบอนุกรมมากกว่าสองไม่ได้ เพียงแต่ว่าเราจะต้องสร้างโปรแกรมควบคุมเอาเอง

ดอสไม่รับรู้ส่วนที่เกินมาจากสอง

ไอบีเอ็มกำหนดหมายเลขของช่องทางเข้าออกสำหรับรีจิสเตอร์ควบคุมของ 8250 และความหมายการใช้งานของรีจิสเตอร์ใน 8250 ดังในตารางที่ 1

สำหรับหมายเลขของทางเข้าออกที่ XF8 และ XF9 ใช้สำหรับ 2 หน้าที่ โดยจะรวมกันเป็นตัวหาร (DLM และ DLL) ถ้าบิตที่ 8 ของ ICR เป็น 1 8250 สุ่มตัวอย่าง (Sampling) สัญญาณที่รับเข้ามา 16 เท่าของอัตราการรับส่งข้อมูล วงจรออสซิลเลเตอร์ในอะแดปเตอร์เองทำความถี่ 1.8432 MHz ป้อนให้กับ 8250 การกำหนดตัวหารเพื่อให้เกิดเป็น Baud Rate ที่ต้องการแสดงอยู่ในตารางที่ 2 ถ้าหากบิตที่ 8 ของ ICR เป็น 0

XF8 จะทำหน้าที่เป็นที่พักของข้อมูลทั้งรับและส่ง (2 รีจิสเตอร์ใช้หมายเลขของช่องทางเข้าออกอันเดียวกัน) ส่วน XF9 จะใช้สำหรับควบคุมการเกิดอินเตอร์รัพต์ โดยมีความหมายของการควบคุมดังแสดงในตารางที่ 3

หมายเหตุ X ในที่นี้หมายถึง 2 สำหรับช่องการสื่อสารที่ 0 และ 3 สำหรับช่องการสื่อสารที่ 1

ตารางที่ 1 ช่องทางเข้าออกและการใช้งานของรีจิสเตอร์ใน 8250

ช่องการสื่อสารที่ 1	ช่องการสื่อสารที่ 2	รีจิสเตอร์	การใช้งาน
3F8	2F8	RBR DLL	- ใช้สำหรับที่พักข้อมูลทั้งรับและส่ง - ใช้เป็นตัวหารตัวที่มีค่าน้อยสำหรับหารความถี่ 1.8432 MHz เป็น Baud Rate
3F9	2F9	DLM IER	- ใช้เป็นตัวหารที่มีค่าสูงสำหรับหารความถี่ 1.8432 MHz ให้เป็น Baud Rate - ใช้สำหรับควบคุมการเกิดอินเตอร์รัพต์
3FA	2FA	IIR	- สำหรับบอกชนิดของอินเตอร์รัพต์
3FB	2FB	LCR	- สำหรับควบคุมพารามิเตอร์การรับส่ง - สำหรับการบ่งบอกการใช้งานของ DLL และ DLM
3FC	2FC	MCR	- ใช้สำหรับควบคุมโมเด็ม
3FD	2FD	LSR	- สำหรับบอกสถานภาพของการรับส่ง
3FE	2FE	MSR	- สำหรับบอกสถานภาพของโมเด็ม

สำหรับรีจิสเตอร์ IIR ที่หมายเลขทางเข้าออก 3FA หรือ 2FA ใช้สำหรับบอกว่าอินเตอร์รัพต์ที่เกิดขึ้นเป็นอินเตอร์รัพต์ชนิดไหน ใน 4 ชนิดที่เราสามารถกำหนดให้เกิดได้โดยรีจิสเตอร์ IER ในโปรแกรมสำหรับจัดการอินเตอร์รัพต์จะต้องมาตรวจสอบรีจิสเตอร์ตัวนี้ก่อนเพื่อที่จะได้จัดการได้ถูกต้องว่าเกิดอินเตอร์รัพต์เพราะเหตุใด การอ่านค่าจากหมายเลขช่องทางเข้าออกนี้จะบอกได้โดยบิตที่ 0 จะเป็นตัวบอกว่าสัญญาณอินเตอร์รัพต์ยังอยู่หรือหายไป (0 ยังอยู่ 1 หายไป) ถ้าหากเกิดมีอินเตอร์รัพต์หลายชนิดซ้อนกัน ชนิดที่มีความสำคัญที่สุดจะปรากฏในรีจิสเตอร์ดังนี้ ตารางที่ 4 บอกชนิดอินเตอร์รัพต์ที่เกิด พร้อมทั้งการกระทำที่จะทำให้สัญญาณอินเตอร์รัพต์หายไป

สาเหตุของการเกิดข้อผิดพลาดในการรับ เกิดขึ้นได้ 3 แบบ

1. เกิดการเขียนทับข้อมูลเก่า คืออ่านข้อมูลออกจาก RBR (หรือรีจิสเตอร์พักข้อมูล) ไม่ทันที่ของใหม่เขียนทับเข้ามา ข้อมูลเก่าจะหายไปเมื่อเกิดเหตุการณ์เช่นนี้ สถานภาพของการรับส่งใน LSR รีจิสเตอร์จะเป็นตัวบอกดังแสดงในรูปที่ 6 พร้อมกับเกิดสัญญาณอินเตอร์รัพต์ไปบอกซีพียู ถ้าเรายอมให้เกิดอินเตอร์รัพต์ชนิดนี้ (ซึ่งกำหนดได้โดยรีจิสเตอร์ LCR ดังที่กล่าวมาแล้ว)

2. เกิดการตรวจสอบผิดพลาดในการส่งข้อมูลเราสามารถแทรกบิตตรวจสอบเอาไว้ทุกอักขระที่ส่ง ถ้าหากฝ่ายรับตรวจสอบแล้วไม่ตรงตามที่ส่งมาฝ่ายรับจะถือว่าเกิดการผิดพลาด LSR รีจิสเตอร์จะเป็นผู้บ่งชี้ และเกิดสัญญาณอินเตอร์รัพต์ทำนองเดียวกับการผิดพลาดในแบบที่ 1

3. เมื่อเกิดการผิดพลาดในการหาขอบเขตของอักขระเรียกว่า Framing Error คือหาบิตเริ่มและบิตหยุดไม่พบ

การควบคุมพารามิเตอร์ของการรับส่งสามารถทำได้โดยการกำหนดค่าลงไป LCR รีจิสเตอร์ซึ่งอยู่ที่หมายเลขช่องทาง XFB โดยค่าใน XFB มีความหมายดังในรูปที่ 4

นอกเหนือจากนั้น 8250 ยังมีรีจิสเตอร์ที่นำไปใช้ในการควบคุมโมเด็มอีกด้วยโดยรีจิสเตอร์ MCR ที่หมายเลขช่องทางเข้าออกที่ XFC ซึ่งมีความหมายการควบคุมดังรูปที่ 5

ตารางที่ 2 ตัวหาร 1.843 MHz เพื่อให้ได้ความถี่ 16 เท่าของ Baud Rate

Baud	ตัวหาร	
	ฐาน 10	ฐาน 16
300	384	180
600	192	0C0
1200	96	060
2400	48	030
4800	24	018
9600	12	00C

ตารางที่ 3 การคุมการกำเนิดสัญญาณอินเตอร์รัพต์ โดยการเขียนข้อมูลเข้าไปที่ช่องทางเข้าออกหมายเลข XF9

บิตที่	0	สร้างสัญญาณอินเตอร์รัพต์เมื่อรับข้อมูลได้ครบ 1 อักขระ
บิตที่	1	สร้างสัญญาณอินเตอร์รัพต์เมื่อรีจิสเตอร์สำหรับส่งว่างลง
บิตที่	2	สร้างสัญญาณอินเตอร์รัพต์เมื่อมีข้อพลาดเกิดขึ้นจากการรับ
บิตที่	3	สร้างสัญญาณอินเตอร์รัพต์เมื่อมีสัญญาณ Modem Status

ตารางที่ 4 รีจิสเตอร์ IIR แสดงชนิดของอินเตอร์รัพต์ที่เกิดขึ้น

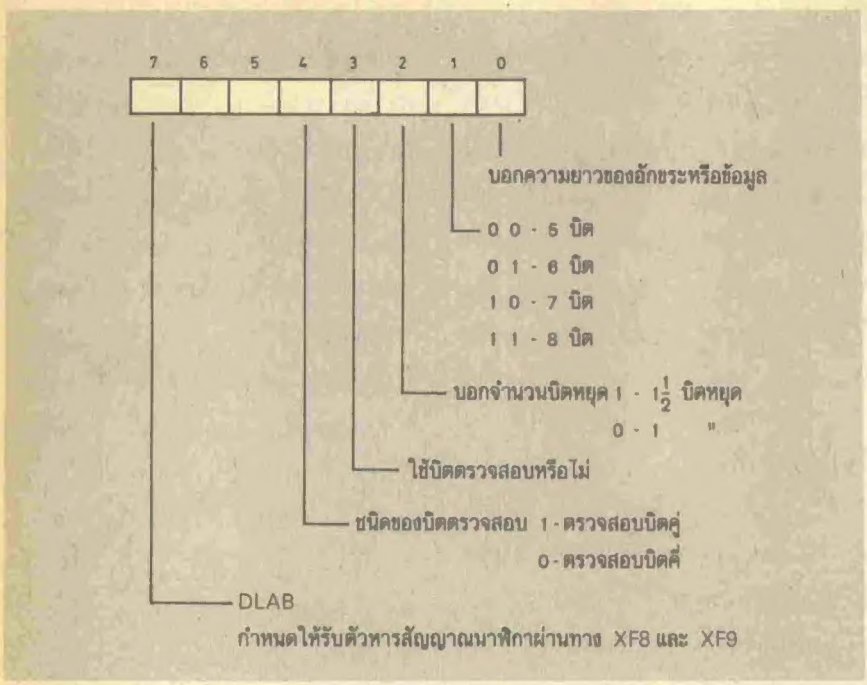
ลำดับความสำคัญ	ค่าที่อ่านได้ใน IIR	สาเหตุของการอินเตอร์รัพต์	ขบวนการทำให้สัญญาณอินเตอร์รัพต์หาย
1	0000110	- เกิดการเขียนทับข้อมูลเก่า - ตรวจสอบ ผิดพลาด - ผิดพลาดของช่วงอักขระ	อ่านรีจิสเตอร์ LSR ที่หมายเลขช่องทาง XFD
2	0000100	- เมื่อรับข้อมูลครบอักขระ	อ่านข้อมูลจาก RBR ที่หมายเลขช่องทาง XF8
3	0000010	- เมื่อส่งข้อมูลออกไปหมด	เขียนข้อมูลลง RBR

โปรแกรมสำหรับควบคุมการรับส่งข้อมูล

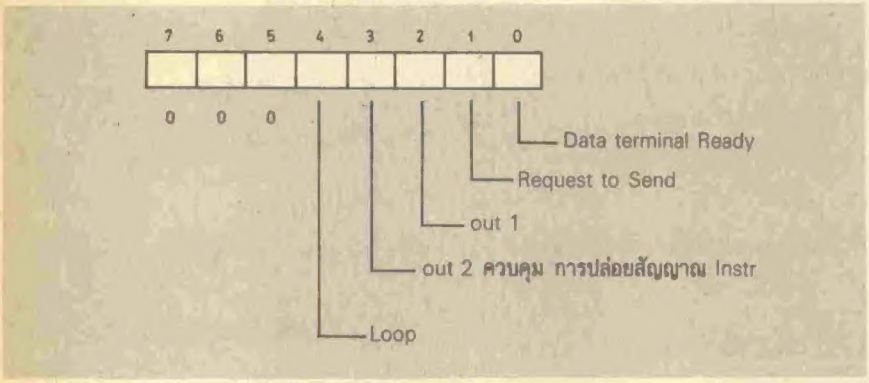
เราได้เรียนรู้โครงสร้างต่าง ๆ ที่จำเป็นสำหรับการสื่อสารข้อมูลของไอบีเอ็มพีซีแล้ว คราวนี้มาลองดูว่าเราจะเขียนโปรแกรมควบคุมการสื่อสารอย่างไรบ้าง โดยไม่กล่าวถึงโปรแกรมสำเร็จรูปที่ใช้ในการสื่อสาร แต่จะกล่าวถึงวิธีการที่จะเขียนโปรแกรมควบคุมการสื่อสารโดยจะเริ่มตั้งแต่ภาษาระดับสูงไปจนถึงระดับที่เราเข้าไปควบคุมเองทุกอย่าง

ลักษณะของการเขียนโปรแกรมการสื่อสาร

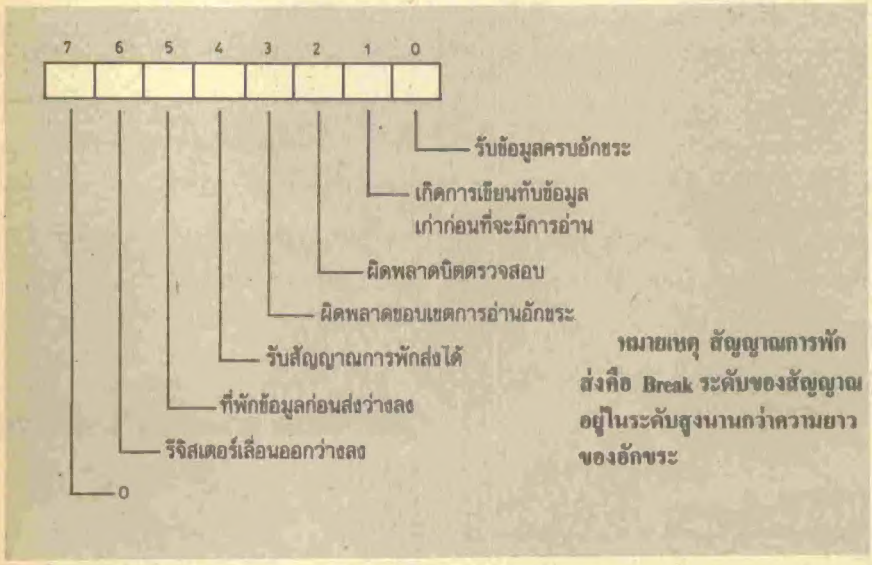
โปรแกรมที่เข้าไปควบคุมการสื่อสารจะต้องรอรับอินพุตอย่างน้อย 2



รูปที่ 4 การตั้งค่าพารามิเตอร์ของการสื่อสารที่หมายเลขช่องทางเข้าออก XFB



รูปที่ 5 การตั้งค่าควบคุมโมเด็ม XFC



รูปที่ 6 สถานภาพของการรับข้อมูล (หมายเลขช่องทางเข้าออกที่ XFD)

ทางที่เกิดขึ้นแบบอสมวาร (Asynchronous) คือไม่ทราบแน่ชัดว่าอินพุตทั้งสองจะเข้ามาเมื่อไหร่ อินพุตทั้งสองทางก็คืออินพุตจากคีย์บอร์ดและจากช่องทางการสื่อสาร วิธีการที่จะรอรับอินพุตจากทั้งสองทาง มีอยู่ 2 วิธีคือ Polling และ การใช้อินเตอร์รัพต์ วิธีการทั้งสองมีข้อดีและข้อเสียต่างกัน

วิธีการหยั่งเสียง (Polling) ก็คือการตรวจสอบอินพุตทั้งสองทางตลอดเวลา ถ้ามีอินพุตจากทางใดก็ให้ไปดำเนินการไปทางนั้นเสียให้เสร็จเรียบร้อยแล้วค่อยตรวจสอบอินพุต ในทางอื่นต่อไป วิธีการเช่นนี้ง่ายต่อการเขียนโปรแกรม แต่มีข้อเสียอยู่บ้างตรงที่มีโอกาสที่จะผิดพลาดมีเยอะถ้าหากโปรแกรมกลับเข้าไปตรวจสอบช่องทางสื่อสารไม่ทัน เพราะข้อมูลเข้าจากช่องทางสื่อสารส่งมาตลอดเวลา ยกตัวอย่างเช่น ในขณะที่ซีพียูใช้เวลาในการเขียนข้อมูลลงดิสค์ซึ่งอาจจะใช้เวลา 4-5 วินาที อาจจะมีข้อมูลเข้ามาที่ช่องทางการสื่อสารมากกว่า 2 อักขระแล้ว ลักษณะเช่นนี้จะทำให้อักขระที่อ่านออกมาไม่ทันหายไป หนทางที่จะแก้ปัญหานี้ก็คือสร้างกฎเกณฑ์ในการติดต่อขึ้นมา (โปรโตคอล) เช่น ให้ฝ่ายที่ต้องการติดต่อส่งอักขระพิเศษ (Request to Send) มาขอการติดต่อก่อน และรอรับคำตอบว่าจะให้ส่งได้หรือไม่จากฝ่ายตรงกันข้าม เนื่องจากอักขระพิเศษที่ส่งมีอักขระเดียว จึงสามารถค้างอยู่ในรีจิสเตอร์พักข้อมูล RBR ของ 8250 ไปจนกว่าซีพียูจะอ่านข้อมูลออกไป ทางฝ่ายรับเมื่อพบว่าคู่สนทนาต้องการติดต่อด้วยที่ให้ส่งอักขระพิเศษอีกตัวหนึ่งตอบกลับไป(Clear to Send)บอกฝ่ายที่เริ่มการติดต่อพร้อมกลับเข้าไปในรูทีนที่ทำหน้าที่ในการรับโดยเฉพาะ ฝ่ายเริ่มการติดต่อหลังจากได้รับสัญญาณ Clear to Send แล้วจึงส่งข้อมูลมาเป็นชุดถ้า

หากเป็นการติดต่อระยะใกล้อาจจะใช้สัญญาณในสาย CTS และ RTS ช่วยแทนที่จะใช้อักขระพิเศษ (ดูรูปที่ 7)

อย่างไรก็ตามการเขียนโปรโตคอลชนิดนี้ต้องคำนึงถึงระยะเวลาที่ต้องคอยคำตอบด้วย ไม่เช่นนั้นฝ่ายหนึ่งอาจจะคอยเก้อ หากฝ่ายตรงข้ามไม่ตอบสนอง

ข้อเสียอีกประการหนึ่งคือ ภาษาที่ใช้ในการเขียนโปรแกรมควบคุมจะต้องมีความเร็วพอที่จะตรวจสอบข้อมูลที่ส่งเข้ามาได้ทัน ตัวอย่างเช่น ภาษาเบสิกถ้าหากใช้เขียนให้เข้าไปตรวจสอบช่องทางสื่อสารโดยตรง ดังในโปรแกรมที่ 1 แล้วจะรับข้อมูลไม่ทันหากความเร็วในการรับส่งมากกว่า 600 baud

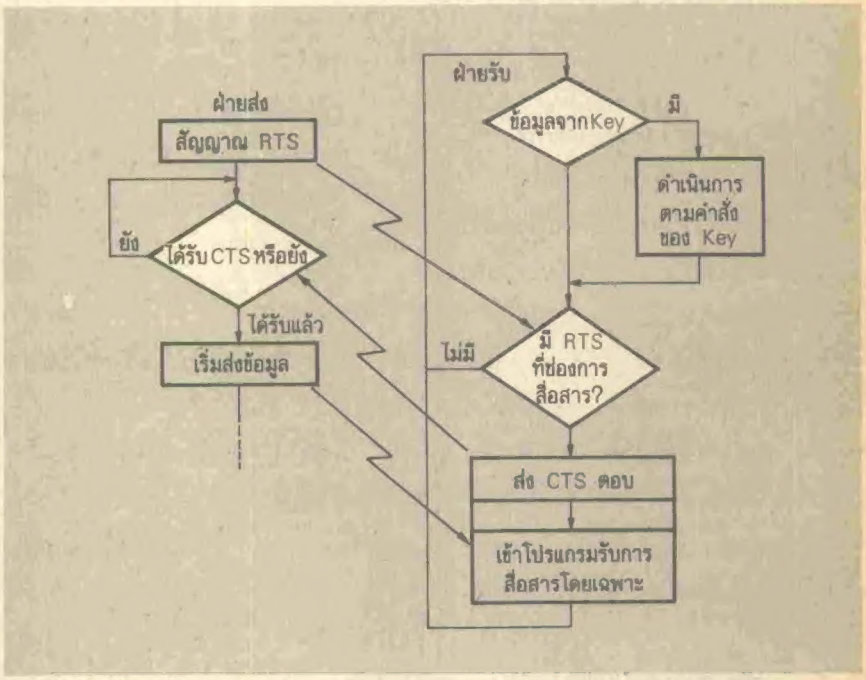
อย่างไรก็ตามเบสิกมีวิธีการที่จะจัดการกับการสื่อสารข้อมูลที่ดีกว่านี้เยอะและเป็นภาษาเดียวที่มีระบบการจัดการเกี่ยวกับการสื่อสารไว้ให้ (ซึ่งจะได้นำมากล่าวต่อไป) ภาษาที่มีความเร็วสูงเช่น ซี หรือเทอร์โบปาสคาลสามารถรับข้อมูลแบบ Polling ได้มากกว่า 9600 baud เท่าที่ผมทดลองมากับปาสคาลและซีความเร็ว B8.4 Kbaud สามารถทำได้สบายสบาย (ตัวอย่างในโปรแกรมที่ 2 และ โปรแกรมที่ 3)

วิธีการเขียนโปรแกรมควบคุมการสื่อสารอีกแบบหนึ่ง คือใช้การขัดจังหวะการทำงานของซีพียู(อินเตอร์รัพต์) เมื่อมีข้อมูลเข้ามาที่ช่องทางการสื่อสารวิธีการเช่นนี้จะทำให้โปรแกรมหลักไม่ต้องพะวงถึง ข้อมูลที่จะเข้ามาทางช่องทางการสื่อสาร เนื่องจากมีโปรแกรมการขัดจังหวะคอยจัดการให้อยู่แล้ว ระบบของไอบีเอ็มพีซีมีระบบการขัดจังหวะสำหรับช่องการสื่อสารไว้พร้อมให้ใช้อยู่แล้วถึงสองช่อง อีกทั้ง 8250 มีระบบการจัดการเกี่ยวกับการขัดจังหวะไว้ให้พร้อมใช้ อย่างไรก็ตามการเขียนโปรแกรมจัดการเกี่ยวกับการขัดจังหวะ (Interrupt Handling Routine) ค่อนข้างจะยุ่งยากและจำเป็นต้องเขียนในรูปแบบของภาษาเครื่องผสมผสานเข้าไปด้วย จึงไม่เหมาะสำหรับผู้เริ่มต้น รายละเอียดของการเขียนอินเตอร์รัพต์จะกล่าวถึงในหัวข้อต่อไป

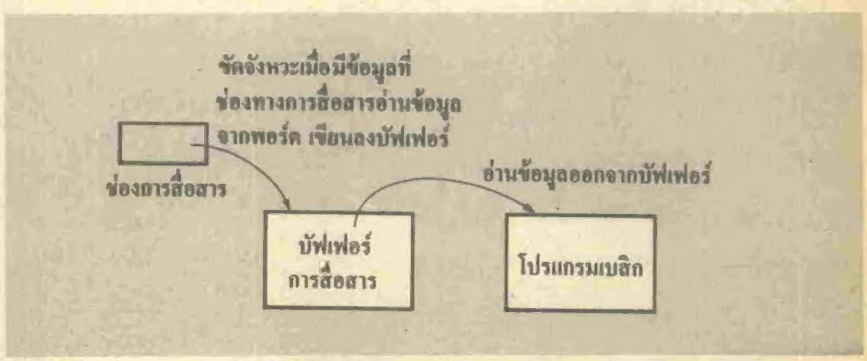
ภาษาเบสิกมีระบบการจัดการเกี่ยวกับการสื่อสารแบบ การขัดจังหวะให้อยู่แล้วโดยที่เราไม่รู้ตัว ทุกครั้งที่เราเปิดช่องทางการสื่อสารแบบอนุกรม โดยเบสิกจะมีบัฟเฟอร์ สำหรับการสื่อสารไว้ให้ 512 ไบต์ ถึง 32 กิโลไบต์ แล้วแต่เราเลือกเอาว่าจะใช้บัฟเฟอร์การสื่อสารมากน้อยแค่ไหน ในลักษณะเช่นนี้ทุกครั้งที่มีข้อมูลเข้ามาที่ช่องทางการสื่อสาร จะมีการขัดจังหวะดึงเอาข้อมูลไปเก็บไว้ในบัฟเฟอร์โดยอัตโนมัติ ถ้าหากเราไม่อ่านข้อมูลออกจากบัฟเฟอร์ก่อน บัฟเฟอร์เต็มเบสิกจะส่ง Error Massage ออกมาเมื่อเราพยายามอ่านออกจากบัฟเฟอร์ว่า "Buffer overflow" เราสามารถกำหนดให้บัฟเฟอร์มีขนาดใหญ่ได้ถึง 32 K โดยตอนแรกเริ่มเรียกเบสิกมาทำงานให้กำหนดขนาดของบัฟเฟอร์การสื่อสารไว้ด้วย เช่น

A > BASIC/C:4096

เป็นการบอกว่าเราต้องการบัฟเฟอร์การสื่อสารขนาด 4 กิโลไบต์ ไม่เช่นนั้นเบสิกจะจองพื้นที่สำหรับเป็นบัฟเฟอร์การสื่อสารแค่ 512 ไบต์ (ดูรูปที่ 8)



รูปที่ 7 การกำหนดโปรโตคอลช่วยในการเขียนโปรแกรมรับส่งแบบ Polling



รูปที่ 8 การทำงานของช่องทางสื่อสารของเบสิก

ภาษาในระดับสูงอื่นๆ ไม่ได้มีคำสั่งที่จะใช้ช่องการสื่อสารอย่างเต็มที่เหมือนเบสิกทุกอย่างจะต้องผ่าน Operating System จัดการให้การเข้าไปควบคุมช่องทางสื่อสารจึงจำเป็นต้องเขียนยูทิลิตี้ขึ้นมาใช้เอง

ยูทิลิตี้เกี่ยวกับช่องทางสื่อสารที่ระบบมีให้ใช้

● **ระดับ Operating System**

ดอสมองช่องการสื่อสารเหมือนกับแฟ้ม ๆ หนึ่ง การจัดการทุกอย่างจึงปฏิบัติเสมือนว่าช่องทางการสื่อสารเป็นแฟ้ม ดอสรู้จักช่องทางการสื่อสารภายใต้ชื่อ AUX:, COM0: และ COM1: Command ของดอสที่เกี่ยวกับช่องการสื่อสารมีดังนี้

1) MODE comn:baud,data,parity, stop,p

Mode เป็น external command เป็นคำสั่งที่อยู่ในชื่อแฟ้ม MODE.COM ใช้สำหรับการกำหนดพารามิเตอร์ของอุปกรณ์รอบนอกต่าง ๆ สำหรับดอสในกรณีนี้ใช้ในการตั้งค่าพารามิเตอร์ของช่องทางการสื่อสาร ตัวอย่างเช่น เราต้องการตั้งค่าพารามิเตอร์ของ COM 0 เป็น 9600 baud ความยาวของอักขระ 8 บิต, ไม่มีบิตตรวจสอบและบิตหยุดเป็น 1 ดังนี้

A > MODE como: 9600,n,8,1,p

P ข้างท้ายหมายถึงให้พยายามติดต่อเรื่อย ๆ ไม่มี Time out

2) CTTY device

เป็นการเปลี่ยน console device จากคีย์บอร์ดและดิสเพลย์มอนิเตอร์ให้ส่งออกไปที่ดีไวซ์แทน เช่น ถ้าเราต้องการจะให้ช่องทางการสื่อสารที่ 1 เข้าไปแทนคีย์บอร์ดและมอนิเตอร์ใช้คำสั่ง

A > CTTY COM1:

จากนั้นอินพุตและเอาต์พุตทุกอย่างจะผ่านช่องการสื่อสารเท่านั้น คีย์บอร์ดและมอนิเตอร์ไม่ได้ใช้อีกต่อไป

3) การเปลี่ยนทิศทางของอินพุตและเอาต์พุตของดอส

ในดอสเราสามารถเปลี่ยนอินพุตและเอาต์พุตของคำสั่งไปออกที่ดีไวซ์หรือแฟ้มไหนก็ได้โดยให้เครื่องหมาย ">", "<", ">>" และ "<<" แทนทิศทาง เช่น

A >DIR > TEXT

แทนที่จะพิมพ์ไดเรคทอรีออกบนหน้าจอก็ไปพิมพ์เก็บไว้ที่แฟ้มชื่อ TEXT

A > DIR >> Text

ต่างจากตัวอย่างข้างบนตรงที่ เอาไปต่อท้ายแฟ้ม TEXT แทนที่จะเขียนทับของเก่า

ทำนองเดียวกัน ถ้าเราต้องการจะเปลี่ยนทิศทางของอินพุต/เอาต์พุตให้ออกไปช่องทางการสื่อสารแทนก็ใส่ชื่อ com0: หรือ com1 แทน เช่น

A > DIR > COM1:

● **บริการสำหรับการสื่อสารของดอส**

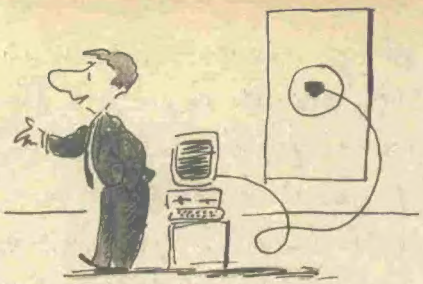
ดอสมีบริการสำหรับการสื่อสารให้เรียกใช้เพียง 2 ฟังก์ชันเท่านั้นคือ ฟังก์ชันที่ 3 และ 4 นอกจากนั้นดอสถือเสมือนว่าช่องการสื่อสารคือแฟ้ม ๆ หนึ่ง ดังนั้น บริการของดอสเกี่ยวกับการอ่านหรือเขียนแฟ้มสามารถนำมาใช้กับช่องทางการสื่อสารได้

ฟังก์ชัน 3 ของดอส (Int 21H, AH = 3)

รับอักขระจาก Auxiliary device (หรือช่องการสื่อสารแบบอนุกรม) มาเก็บไว้ในรีจิสเตอร์ AL

ฟังก์ชันที่ 4 ของดอส (Int 21H, AH = 4)

ส่งอักขระใน AL ออกไปที่ Auxiliary device



● **บริการสำหรับการสื่อสารของไบออส (BIOS)**

ไบออสมีบริการเกี่ยวกับการสื่อสารให้เลือกใช้คือ อินเตอร์รัพต์ที่ 14H โดยมีฟังก์ชันให้ 3 อัน คือ

AH = 0 สำหรับตั้งค่าพารามิเตอร์

AH = 1 ส่งอักขระ AL ออกไปที่ช่องทางออกอนุกรม

AH = 2 รับอักขระจากพอร์ตอนุกรมเข้ามาที่ AL

(ดูรายละเอียดในเรื่องของไบออสในไมโครฯ ฉบับที่ 30 ประจำเดือนมิถุนา 30)

การพัฒนายูทิลิตี้สำหรับการสื่อสารข้อมูล

ในภาษาระดับสูงหลายภาษาที่ไม่มีคำสั่งเกี่ยวกับการควบคุมช่องทางสื่อสารให้ หากเราจำเป็นต้องใช้คอมพิวเตอร์ให้ติดต่อกับโลกภายนอกทางช่องทางการสื่อสาร จำเป็นที่เราจะต้องพัฒนายูทิลิตี้ขึ้นมาใช้เอง ในที่นี้จะกล่าวถึงเฉพาะการเขียนอินเตอร์รัพต์รูทีนสำหรับช่องการสื่อสารเพื่อใช้กับภาษาซีและเทอร์โบปาสคาลพอเป็นแนวทางเท่านั้น

ฮาร์ดแวร์อินเตอร์รัพต์

ไอบีเอ็มพีซีมีสัญญาณการขัดจังหวะที่สามารถเข้าไปขัดจังหวะการทำงานของซีพียูได้ 8 ช่องทาง โดยใช้ไอซี 8259 สำหรับควบคุมการขัดจังหวะ ไอบีเอ็มพีซีได้กำหนดการใช้งานของการขัดจังหวะจากฮาร์ดแวร์ทั้ง 8 ไว้ดังนี้

IRQ0 สำหรับการขัดจังหวะของ Timer

IRQ1 สำหรับการขัดจังหวะของคีย์บอร์ด

IRQ2 สำรอง

IRQ3 สำหรับช่องการสื่อสารหมายเลข 1

IRQ4 สำหรับช่องการสื่อสารหมายเลข 0

IRQ5 สำหรับดิสเกตต์คอนโทรลเลอร์

IRQ6 สำหรับดิสก์คอนโทรลเลอร์

IRQ7 สำรอง

สัญญาณจากการขัดจังหวะทั้ง 8 จะทำให้เกิดอินเตอร์รัพต์ที่ 8 บวกหมายเลขของ IRQ ดังนั้นการขัดจังหวะจาก com0 ซึ่งผ่านทาง IRQ4 จึงทำให้เกิดทิศทางการขัดจังหวะที่ OCH หรือที่ 13 ในเลขฐาน 10

ขั้นตอนของการยอมให้สัญญาณการขัดจังหวะผ่านเข้าไปถึงซีพียูแสดงในรูปที่ 9

รัพต์จากช่องการสื่อสารอธิบายได้ดังนี้

1. เมื่อมีเหตุการณ์เกิดขึ้นที่ 8250 ตามที่เรากำหนดเหตุการณ์ที่จะทำให้เกิดสัญญาณอินเตอร์รัพต์ที่พอร์ต XF9 เช่น ตั้งค่าไว้ที่ XF9 เป็น 01 สัญญาณ Instr จะออกมาจาก 8250 เมื่อ 8250 รับข้อมูลครบ 1 อักขระ

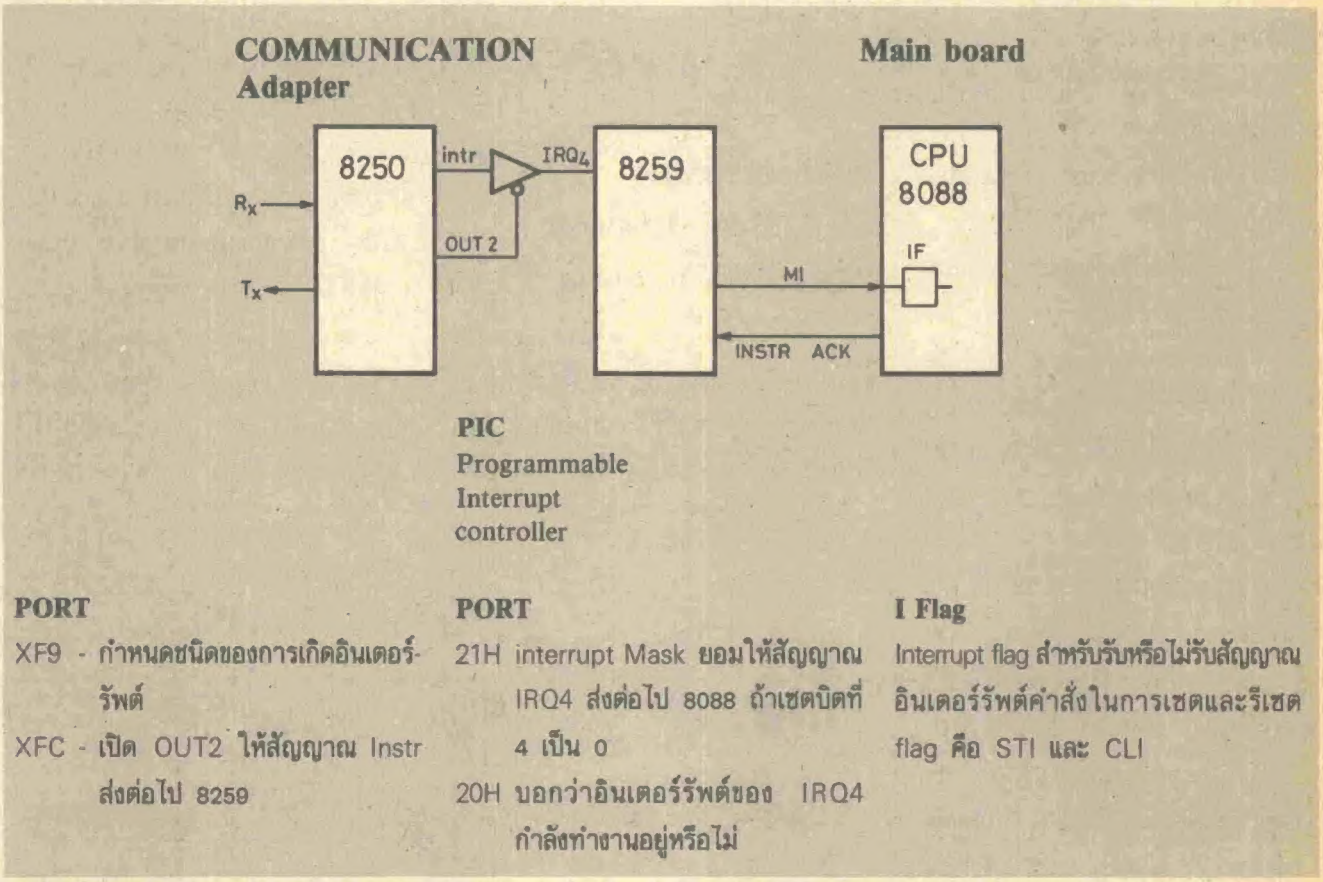
2. สัญญาณ Instr ที่ออกมาจาก 8250 ยังถูกควบคุมด้วยเกตตัวหนึ่งซึ่งสามารถควบคุมได้โดยพอร์ต XFC บิตที่ 3 (สัญญาณ OUT 2)

3. เมื่อ 8259 รับสัญญาณ IRQ4 จะตรวจสอบดู IMR หรือ Interrupt Mask Register ว่าจะรับหรือไม่ 8259 จะรับสัญญาณ IRQ ต่อเมื่อบิตที่ตรงกับหมายเลข IRQ ใน IMR รีจิสเตอร์เป็น 0 เท่านั้น เราเข้าไปควบคุม IMR ได้โดยช่องทางเข้าออกหมายเลข 21H (พอร์ต = 21H)

4. ถ้า 8259 รับสัญญาณ IRQ 8259 จะตรวจสอบลำดับความสำคัญของสัญญาณ IRQ ว่าขณะนี้ซีพียูกำลังทำงานให้กับอินเตอร์รัพต์ไหนบ้าง โดยตรวจสอบดูที่ รีจิสเตอร์ ISR หรือ Interrupt Service Register ถ้าซีพียูไม่ได้ให้บริการอินเตอร์รัพต์ไหนที่สำคัญกว่า 8259 จะส่งสัญญาณอินเตอร์รัพต์ต่อไปยัง 8088 ผ่านทาง Maskable Interrupt (MI)

5. ทุก ๆ คำสั่งของ 8088 หลังจากปฏิบัติแล้ว จะตรวจดูสัญญาณอินเตอร์รัพต์ที่ขา NMI (Non Maskable Interrupt ซึ่งไอบีเอ็มพีซี ต่อเอาไว้กับการตรวจความผิดพลาดของหน่วยความจำ) และ MI สำหรับสัญญาณ MI จะมีผลให้ 8088 ให้ความสนใจต่อเมื่ออินเตอร์รัพต์แฟลกเป็น 1

6. เมื่อ 8088 รับสัญญาณ MI จะส่งสัญญาณไปบอก 8259 ว่า ตกลงรับการร้องขออินเตอร์รัพต์ (Instr. Ack) กลับไปยัง 8259



รูปที่ 9 ขั้นตอนการเดินทางของสัญญาณการขัดจังหวะจากช่องทางสื่อสารอนุกรม

7. 8259 จะเอาหมายเลขทิศทางการขัดจังหวะ (OCH ในที่นี้) ส่งให้ 8088 ผ่านทางดาต้าบัส พร้อมกับตั้งค่า ISR บิตที่ตรงกับหมายเลขสัญญาณ IRQ ให้เป็น 1 เพื่อเป็นการเตือนตัวเองว่าขณะนี้ซีพียูกำลังให้บริการอินเตอร์รัพต์ไหนบ้าง

8. 8088 รับเอาหมายเลขทิศทางการขัดจังหวะจากดาต้าบัสแล้วจะเก็บค่าภาวะแฟลกและเซกเมนต์ของโปรแกรมและโปรแกรมเคาน์เตอร์ขณะนั้นลงในสแต็กพร้อมกับกระโดดไปยังอินเตอร์รัพต์รูทีนที่ชี้บอกโดย Interrupt Vector Table (ในหน่วยความจำ ตำแหน่งที่ 0000: 0030H ในที่นี้) พร้อมกับเคลียร์ IF flag ให้เป็น 0 ไม่ยอมให้เกิด Maskable Interrupt อีกต่อไป

9. เป็นหน้าที่ของอินเตอร์รัพต์รูทีนที่จะจัดต่อไป

หมายเหตุ สัญญาณ Instr ของ 8250 จะหายไปเมื่อมีการอ่านรีจีสใน 8250 ตามตารางที่ 3

เทคนิคการเขียนอินเตอร์รัพต์รูทีนสำหรับการสื่อสาร

1) ตั้งค่าพารามิเตอร์ต่าง ๆ สำหรับการสื่อสาร

2) กำหนดชนิดของการเกิดสัญญาณอินเตอร์รัพต์ใน 8250

3) กำหนด Vector table สำหรับการอินเตอร์รัพต์ของการสื่อสารที่จะเข้ามา (ทิศทางที่ 0CH สำหรับ IRQ4 หรือ com0: และ 0BH สำหรับ IRQ3 หรือ com1:)

4) ตั้งค่า IMR ใน 8259 โดยผ่านทางพอร์ต 21H

5) เปิดอินเตอร์รัพต์เกตใน 8250 โดยให้บิตที่ 3 ของ MCR เป็น 1

6) ต้องไม่ลืมนำอินเตอร์รัพต์เวคเตอร์เก่ากลับคืน เมื่อเราเอาอินเตอร์รัพต์รูทีนออกจากหน่วยความจำหรือเลิกใช้โปรแกรมนี้แล้ว

ตัวรูทีนที่ให้บริการสำหรับการขัดจังหวะสำหรับช่องทางสื่อสารควรจะมีลักษณะดังนี้

1) เซตค่า IF flag ให้เป็น 1 กลับคืนถ้าหากต้องการให้เกิด Maskable interrupt ตัวอื่นที่มีความสำคัญกว่า

2) เก็บค่ารีจิสเตอร์ต่าง ๆ ที่ต้องการใช้ลงในสแต็กให้หมด

3) อ่านดูรีจิสเตอร์ IIR ใน 8250 ว่าเป็นการเกิดอินเตอร์รัพต์ด้วยเหตุใดในกรณีที่เราตั้งค่าให้เกิดอินเตอร์รัพต์หลายชนิด

4) อ่านอินพุตมาจาก 8250 และจัดการตามขบวนการ

5) เอาค่ารีจิสเตอร์ต่าง ๆ ที่เก็บไว้ในสแต็กกลับคืน

6) ส่งเอาต์พุตไปบอก 8259 ว่าสิ้นสุดการทำงานอินเตอร์รัพต์ที่ขอมาแล้ว โดยส่งไปที่พอร์ต 20H ด้วยค่า 20H บวกหมายเลข IRQ ที่ 8259 รับเข้ามา เช่น ถ้าเป็น IRQ4 ก็ให้ส่งค่า 24H ออกไปที่พอร์ต 20H

7) IRET กระโดดกลับไปโปรแกรมหลัก

ตัวอย่างการเขียนอินเตอร์รัพต์ โปรแกรมสำหรับเทอร์โบปาสคาล

โปรแกรมที่ 4 แสดงถึงการเขียนฮาร์ดแวร์อินเตอร์รัพต์รูทีนที่ใช้สำหรับการสื่อสาร โดยตัวโปรแกรมหลักไม่ได้ทำอะไรเพียงแต่ Loop พิมพ์ข้อความเล่น ๆ เพื่อแสดงให้เห็นว่าโปรแกรมหลักไม่ต้องพะวงว่า ข้อมูลที่จะเข้ามาทางช่องสื่อสารนั้นจะมาเมื่อไหร่ เพียงแต่คอยดูว่าเมื่อไหร่โปรแกรมการขัดจังหวะจะรับข้อมูลครบบรรทัด เพื่อจะได้นำข้อมูลนั้นออกมาพิมพ์

หลักการเขียนโปรแกรมการขัดจังหวะก็คือ ทำอย่างไรจึงจะเชื่อมโยงตำแหน่งที่อยู่ของตัวแปรในโปรแกรมขัดจังหวะและโปรแกรมหลักเข้าหากันได้ เนื่องจากจากการขัดจังหวะอาจจะเกิดเมื่อไหร่ก็ได้ ซึ่งในขณะนั้นรีจิสเตอร์ DS อาจจะไม่ได้อยู่ในเซกเมนต์เดียวกันกับเทอร์โบปาสคาล ดังนั้นเราจึงจำเป็นต้องเก็บเซกเมนต์ของ DS ไว้สักที่ ซึ่งจะต้องอยู่ในเซกเมนต์เดียวกันกับโค้ดเซกเมนต์ของตัวโปรแกรมบริการการขัดจังหวะอยู่ ซึ่งก็คือโค้ดเซกเมนต์ของเทอร์โบปาสคาล เนื่องจากว่าเราเขียนโปรแกรมบริการการขัดจังหวะในเทอร์โบปาสคาล บอร์แลนด์ยืนยันว่า ตำแหน่งที่ 5 ถึงตำแหน่งที่ 10 ในโค้ดเซกเมนต์ไม่ได้ใช้งานเราสามารถใช้เก็บข้อมูลได้โดยไม่รบกวนการทำงานของเทอร์โบ จะเห็นว่าในบรรทัดที่ 13 ของโปรแกรม COMINT .PAS จะสำแดงตัวแปรที่ชื่อ dsave เอาไว้ที่โค้ดเซกเมนต์ตำแหน่งที่ 6

การเขียนโปรแกรมบริการการขัดจังหวะ (ชื่อ Procedure Interrupt ในที่นี้) จะทำการเขียนส่วนหัวและส่วนหางเป็นภาษาเครื่องโดยใช้

Inline statement ของเทอร์โบ ส่วนหัวทำหน้าที่เก็บรีจิสเตอร์ต่าง ๆ ลงสแต็กไว้ก่อน เราไม่ทราบแน่ชัดว่าเทอร์โบจะใช้รีจิสเตอร์ตัวไหนบ้าง ฉะนั้นทางที่ดีก็เก็บรีจิสเตอร์ทุกตัวลงสแต็กเอาไว้ จากนั้นก็ต้องปรับค่า Data Segment Register (DS) ให้ตรงกับที่เทอร์โบใช้งาน (บรรทัดที่ 31 ถึง 35) โดยดึงเอาค่า DS ที่เก็บเอาไว้ตอนเริ่มแรก

การทำงานของ Main Program (บรรทัดที่ 95)

ตัวของโปรแกรมบริการการขัดจังหวะเขียนเป็นภาษาเทอร์โบปาสคาล (บรรทัดที่ 36-47) โดยอ่านข้อมูลมาจากที่พักข้อมูลของ 8250 จับไปเก็บไว้ในสตริงที่ชื่อว่า buffer โดยพอยน์เตอร์เป็นตัวบอกว่าควรจะนำไปเก็บไว้เป็นตัวที่เท่าไหร่ เนื่องจากเทอร์โบเก็บสตริงเป็นลักษณะของไบต์อะเรย์โดยมีตัวแรกของอะเรย์บอกว่าสตริงนี้ยาวเท่าไหร่ ขณะที่รับอักขระมาจากช่องทางการสื่อสารก็ตรวจสอบดูว่าเป็น Carriage Return หรือเปล่า ถ้าใช่ก็ยกแฟลกบอกว่ารับข้อมูลครบหนึ่งบรรทัดแล้ว

ส่วนท้ายของโปรแกรมบริการการขัดจังหวะต้องนำค่ารีจิสเตอร์ต่าง ๆ ที่เก็บเอาไว้กลับคืนมา (บรรทัดที่ 48 ถึง 55) ก่อนที่จะออกจากโปรแกรมบริการขัดจังหวะ (IRET)

ต้องไม่ลืมว่า เราจะต้องส่งข้อความไปบอก 8259 ว่าโปรแกรมการขัดจังหวะได้สิ้นสุดลงแล้ว (บรรทัดที่ 46) เพื่อให้ 8259 ทราบสถานภาพการทำงานของซีพียูเกี่ยวกับการบริการการขัดจังหวะ

Procedure Set-IVT (บรรทัดที่ 58-68) ทำหน้าที่เปลี่ยนตารางทิศทางการขัดจังหวะ (Interrupt Vector Table) ชี้ไปที่โพรซีเดอร์อินเตอร์รัพต์ ซึ่งเราตั้งใจว่าจะให้เป็นโปรแกรมบริการการขัดจังหวะโดยในตารางนั้นจะต้องมีโค้ดเซกเมนต์ 2 ไบต์ และออฟเซต 2 ไบต์ ซึ่งเป็นที่อยู่ของโปรแกรมบริการขัดจังหวะตำแหน่งของตารางเริ่มต้นในหน่วยความจำที่เซกเมนต์ 0, ออฟเซตที่หมายเลขการขัดจังหวะคูณด้วย 4

โปรดสังเกตด้วยว่าทำไมเราจึงเอาออฟเซตของโพรซีเดอร์อินเตอร์รัพต์บวกกับ 7 (บรรทัดที่ 62) เป็นออฟเซตที่จะต้องใส่ลงในตารางทิศทางการขัดจังหวะ ทั้งนี้เนื่องจากว่าเมื่อเทอร์โบปาสคาล คอมพายล์ คำสั่ง inline ออกมาเป็นแมกชีนโค้ดนั้น จะใช้พื้นที่ 7 ไบต์เป็นส่วนหัว ดังนั้นที่ตำแหน่ง ofs (Interrupt) จะไม่ใช่คำสั่ง STI ดังที่เราเขียนเอาไว้ โดยคำสั่ง Inline

Procedure Set-com (บรรทัดที่ 69-82) เป็นการตั้งค่าความเร็วของการรับส่งโดยการเข้าไปควบคุม 8250 โดยตรง

Procedure Enable-IRQ เป็นการตั้งค่า IMR ใน 8259 ให้ยอมรับสัญญาณอินเตอร์รัพต์จาก 8250 (หรือฮาร์ดแวร์อินเตอร์รัพต์ตัวอื่น)

ตัวอย่างการเขียนอินเตอร์รัพต์รูทีนสำหรับซี

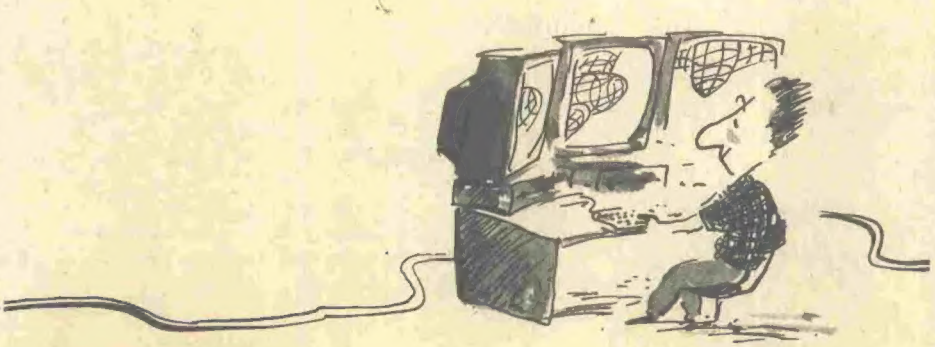
เนื่องจากซีเป็นภาษามาตรฐานที่สามารถเรียกใช้รูทีนที่เขียนโดยแอสเซมบลีได้ ดังนั้นจึงเป็นการง่ายที่จะสร้างยูทิลิตี้ใด ๆ บนภาษาเครื่องแล้วนำมาใช้กับซี

ตัวอย่างในโปรแกรมที่ 5 เป็นรูทีนที่เขียนโดยภาษาแอสเซมบลีสำหรับจัดการเรื่องฮาร์ดแวร์อินเตอร์รัพต์โดยมีโพรซีเดอร์อยู่ 2 อันคือ Hint สำหรับตั้งค่าฮาร์ดแวร์อินเตอร์รัพต์ และ ICOM 0 สำหรับเป็นโปรแกรมบริการการขัดจังหวะ การเชื่อมต่อกับ C ทำได้โดยการสำแดงตัวแปรที่ชื่อ rxbuff และ rflag ให้เป็นที่รู้กันในทุกโมดูลที่จะ link เข้าหากัน

โปรแกรมที่ 6 เป็นโปรแกรมรับส่งข้อมูลที่สามารถนำไปใช้ทดสอบรับและส่งได้เลย โดยทางด้านการรับจะปล่อยให้โปรแกรมบริการการขัดจังหวะจัดการให้ ขณะที่โปรแกรมหลักคอยรับข้อมูลจากคีย์บอร์ดสำหรับที่จะส่งออกทางช่องทางการสื่อสาร

บทส่งท้าย

การที่จะนำไมโครคอมพิวเตอร์ไปใช้ให้เป็นประโยชน์มากขึ้น จำเป็นจะต้องให้การสื่อสารกับไมโครคอมพิวเตอร์ งานควบคุมทุกชนิดสามารถนำไมโครคอมพิวเตอร์ให้ใช้ควบคุมได้ แต่จะต้องทำให้ไมโครคอมพิวเตอร์สามารถรับข้อมูลจากโลกภายนอกที่นอกเหนือจากคีย์บอร์ดเสียก่อน การศึกษาเกี่ยวกับการสื่อสารข้อมูลของไมโครคอมพิวเตอร์จึงเปรียบเหมือนการเปิดแนวทางใหม่ในการที่จะนำไมโครคอมพิวเตอร์มาประยุกต์ใช้งาน



โปรแกรม 1 การใช้เบสิก Polling ข้อมูลเข้า

```
 5 REM  DEMO PROGRAM FOR POLLING RECEIVE COMMUNICATION
10 OPEN "com0:300,n,8,1,cs,ds" as #1 'SET BUAD RATE AND PARAMETER
20 CLOSE #1                          'IF DON'T INPUT AS A FILE
30 CLS
40 STAT=INP(&H3FD)                   'GET LINE STATUS
50 IF (STAT AND 1)!=1 THEN GOTO 40   'IF NO DATA RECEIVED THEN GOTO 40
60 C=INP(&H3F8)                      'IF SO GET CHAR FROM DATA PORT
70 A$=A$+CHR$(C)
80 IF C=13 THEN PRINT A$;:A$=""      'IF CARRIAGE RETURN IS RECEIVED
90 GOTO 40
```

โปรแกรม 2 การใช้เทอร์โบปาสคาลสำหรับ Polling ข้อมูลเข้า

```
Program prog2;
 {
  Demo Program how to polling input data from com. port
  Written by Dr. Paisan Sangaunmu 12-07-87

  The main program is scan the line status if there is indicator
  that a character is received then get it and put it in the buffer
  if carriage return is received then print it.
  }
var
   buffer   :string[127];
   c,flag   :byte;
   rxptr    :integer;

Procedure Set_com;
var
  sts       :byte;
begin
      sts       :=port[$3fb];       (status of the 8250)
      port[$3fb] :=sts+128;
      port[$3f9] :=0;
      port[$3f8] :=$60;             (set baud rate to 1200)
      port[$3fb] :=sts;
      port[$3f9] :=01;              (Interrupt when Data Ready)
      sts       :=port[$3f8];       (clear any input before)
      port[$3fc] :=$0a;             (Turn interrupt gate)

end;

begin
   set_com;
   rxptr:=1;
   clrscr;
   while (1=1) do
     begin
     flag:=0;
      while(flag=0) do              {   scan line status     }
        begin                       {   any character receive?}
          flag     :=port[$3fd];
          flag     :=flag and 1;
        end;
       c          :=port[$3f8];     {  If so, get the character }
      buffer[rxptr]:=chr(c);        (and put it into the buffer )
      rxptr      :=rxptr+1;
      if c=13 then                  (If carriage return received )
        begin
          buffer[0]:=chr(rxptr);    (set the length of the string)
          writeln(buffer);          (and print it               )
          rxptr:=1;                 (Reset the receive poiter   )
        end;
    end;
  end.
```

โปรแกรม 3 การใช้ซี Polling ข้อมูลเข้า

```
/*************************************************
   DEMO PROGRAM HOW THE COMMUICATION PROGRAM
   USING THE POLLING METHOD

   COMPILE THIS PROGRAM BY THE MICROSOFT C VERSION 1
   A>MC1 PROG3
   A>MC2 PROG3

   THEN LINK THIS PROGRAM
   A>LINK C+PROG3,PROG3,,MC

   DR.PAISAN SANGUANMU 12-07-87
*************************************************/

#include <stdio.h>
char rxbuff[1024];   /*Communiction buffer  1 k byte*/

main()

{
int stat,i,c,ptr;

 /*set communication buad rate */
 stat =inp(0x3fb);
                                /*set bit #7 to 1 */
 outp(0x3fb,stat | 128);        /*then write to LCR */
 outp(0x3f8,0x0c);              /*set baud rate to 9600*/
 outp(0x3f9,00);
 outp(0x3fb,stat);              /*restore old defualt parameter*/
 printf("Waiting for communication \n ");
 while(1)
  {
    /*check communication port and the key board*/
    if((inp(0x3fd)&1)==1)     /*any character at coma. port?       */
      {
       c=inp(0x3f8);          /*read character from coma. port if so*/
       rxbuff[ptr]=c;         /*put it in the buffer       */
       ptr++;
       if(c==13)                    /*is it a carriag return?   */
         {rxbuff[ptr]=0;            /*terminate the string      */
          printf("%s",&rxbuff[0]);/*print the received string */
          ptr=0;                    /*reset receive pointer     */
         }
      }
    if(bdos(11,0)==1)               /*any key struck?           */
       if(bdos(8,0)==27)break;      /*if Esc. is hit then exit  */
   }
   exit(0);
 }
```

โปรแกรม 4 การเขียนฮาร์ดแวร์อินเตอร์รัพต์ สำหรับเทอร์โบปาสคาล

```
{
 Demo Program how to use hardware interrupt for communication
 program.
 Written by Dr. Paisan Sangaunmu 12-07-87

 The main program is doing nothing but printing the waiting
 massage without worrying about the data coming at the communication
 port. The interrupt routine will raise the flag when a carriage
 return is recieved.
 }
Program Comint(input,output);
var
```

```
13:       dsave  :integer absolute Cseg:$0006;
14:       buffer :string[127];
15:       txptr  :integer;
16:       rxptr  :byte;
17:       c      :byte;
18:       flag,x :integer;
19:   Procedure Interrupt;
20:   begin
21:    {save register flag}
22:    inline($FB/        {STI       enable further int.        }
23:           $1E/        {PUSH DS                              }
24:           $50/        {PUSH AX                              }
25:           $53/        {PUSH BX                              }
26:           $51/        {PUSH CX                              }
27:           $52/        {PUSH DX                              }
28:           $56/        {PUSH SI                              }
29:           $57/        {PUSH DI                              }
30:           $06);       {PUSH ES                              }
31:    inline($8C/$C8/    {MOV AX,CS   restore data segment     }
32:           $8E/$D8/    {MOV DS,AX   allow us to access absolute addr.}
33:           $A1/dsave/  {MOV AX,dsave                         }
34:           $8E/$D8);   {MOV DS,AX   now DS is in PASCAL      }
35:
36:    {Main body of the interrupt routine}
37:     c:=port[$3f8];
38:     buffer[rxptr]:= chr(c);
39:     rxptr:=(rxptr+1);    {limit size to 127}
40:     if(c=13) then
41:       begin
42:         flag:=1;
43:         buffer[0]:=char(rxptr);
44:         rxptr:=1;
45:       end;
46:     port[$20]:=$24;                 {non specific EOI to 8259}
47:
48:     inline($07/        {POP ES     }
49:            $5F/        {POP DI     }
50:            $5E/        {POP SI     }
51:            $5A/        {POP DX     }
52:            $59/        {POP CX     }
53:            $58/        {POP BX     }
54:            $58/        {POP AX     }
55:            $1F/        {POP DS     }
56:            $CF);       {IRET       }
57:   end;
58:   Procedure Set_IVT(entry:byte);
59:   var
60:      offset,segment,fword,sword:integer;
61:   begin
62:      offset   :=Ofs(Interrupt)+7;
63:      segment  :=Cseg;
64:      fword    :=(entry+8)*4;
65:      sword    :=fword+2;
66:      mem[$0000:fword] :=offset;
67:      mem[$0000:sword] :=segment;
68:   end;
69:   Procedure Set_com;
70:   var
71:     sts       :byte;
72:   begin
73:       sts       :=port[$3fb];       {status of the 8250}
74:       port[$3fb] :=sts+128;
75:       port[$3f9] :=0;
76:       port[$3f8] :=$60;             {set baud rate to 1200}
77:       port[$3fb] :=sts;
78:       port[$3f9] :=01;              {Interrupt when Data Ready}
79:       sts       :=port[$3f8];       {clear any input before}
80:       port[$3fc] :=$0a;             {Turn interrupt gate}
81:
82:   end;
83:
84:   Procedure Enable_IRQ(IRQ:byte);
85:   var
86:     imr,mask   :byte;
87:   begin
88:      mask      :=not(1 shl IRQ);
89:      imr       :=port[$21];       {get IMR from 8259}
90:      imr       :=imr and mask;
91:      port[$21] :=imr;
92:   end;
93:
94:   begin
95:      dsave :=Dseg;
96:      set_IVT(4);
97:      enable_irq(4);
98:      set_com;
99:      flag:=0;
100:     rxptr:=1;
101:     clrscr;
102:     while (1=1) do
103:       begin
104:       x:=0;
105:       while(flag=0) do
106:         begin
107:          gotoxy(10,10);
108:          writeln('waiting for communication  ',x);
109:          x:=x+1;
110:         end;
111:       gotoxy(1,20);
112:       write(buffer);
113:       flag:=0;
114:     end;
115:   end.
116:
```

โปรแกรม 5 Assemble โปรแกรมสำหรับฮาร์ดแวร์อินเตอร์รัพต์ใช้ร่วมกับซี

```
1:  ;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2:  ;DEMO ROUTINE FOR COMMUNICATION HARDWARE INTERRUPT
3:  ;ASSEMBLE THIS PROGRAM
4:  ;A>MASM COMINT,,,,
5:  ;TO GET THE COMINT.OBJ
6:  ;THEN LINK THIS OBJ
7:  ;WITH THE MICROSOFT C (VERSION 1) PROGRAM
8:  ;
9:  ;DR.PAISAN SANGUANMU 12-07-87
10:
11: DGROUP  GROUP DATA
12: DATA    SEGMENT BYTE PUBLIC 'DATA'
13:         PUBLIC  RXBUFF,RFLAG
14:
15:         PORT    EQU 3F8H            ;BASE COMM PORTO
16:         RXBUFF  DB  128 DUP(0)      ;DEFUALT RECEIVE BUFFER
17:                 DB  0
18:         RW_OFS  DW  0
19:                 DW  0
20:         RFLAG   DW  0
21: DATA    ENDS
22: PGROUP  GROUP PROG
23: PROG    SEGMENT BYTE PUBLIC 'PROG'
24:         PUBLIC HINT
25: EOI     EQU 24H             ;END OF INTERRUPT FOR IRQ4
26:         ASSUME CS:PGROUP,DS:DGROUP
27:        ;SET HARDWARE INT.
28:
29: ;INTERRUPT ROUTINE FOR COMM 0
30: ICOM0 PROC NEAR
31:        ;SAVE ALL REGISTER
32:         STI                 ;SET INT MASK BIT ON  ,NO MORE INT
33:         PUSH DS
34:         PUSH AX
35:         PUSH BX
36:         PUSH CX
37:         PUSH DX
38:         PUSH DI
39:         PUSH SI
40:         PUSH BP
```

```
RX_INT:
        MOV AX,DGROUP
        MOV DS,AX
        MOV DX,PORT         ;COMM PORT
        IN  AL,DX           ;READ CHAR FROM 8250 PORT
        AND AL,7FH          ;MASK BIT #8
        MOV BX,OFFSET RXBUFF
        MOV DI,RW_OFS       ;where is the last read
        MOV [DI+BX],AL      ;SAVE INCOM CHARACTER IN THE BUFFER
        INC DI
        AND DI,07FH         ;WRAP AROUND AT 128 BYTE
        MOV RW_OFS,DI       ;RESTORE WRITE POINTER
        CMP AL,13           ;IS IT CR?
        JNE IO_EXIT
        MOV RW_OFS,0        ;IF SO RESET RW POINTER
        MOV RFLAG,1         ;SET RECEIVED FLAG
IO_EXIT:
        MOV AL,EOI          ;END OF INTERRUPT SERVICE
        OUT 20H,AL
        POP BP
        POP SI
        POP DI
        POP DX
        POP CX
        POP BX
        POP AX
        POP DS
        IRET
ICOMO ENDP
;$$$$SET INTERRUPT VECTOR TABLE FOR IRQ4 OR TABLE ENTRY #0CH$$$$$$$
HINT PROC  NEAR
     ;SET COM(1) INTERRUPT ROUTINE
     PUSH BP
     MOV  BP,SP
     MOV  DX,OFFSET ICOMO  ;INTERRUPT ENTRY POINT
     MOV  AX,250CH         ;FUNCTION 25H FOR SET INT. VECTOR
     PUSH DS
     PUSH CS               ;LOAD PROGRAM SEGMENT
     POP  DS               ;SEGMENT OF THE INTERRUPT ROUTINE
     INT  21H              ;LET DOS CHANGE THE INTERRUPT VECTOR
     POP  DS
     ;ENABLE INTERRUPT MASK OF 8259
     IN   AL,21H           ;READ MASK REGISTER OF 8259
     AND  AL,11101111B     ;RESET MASK FOR IRQ 4
     OUT  21H,AL
     ;RESET ANY INTERRUPT IN SERVICE FOR IRQ 4
     MOV  AL,EOI
     OUT  20H,AL           ;SEND END OF INTERRUPT TO 8259
     POP  BP
     STI                   ;SET INT ON
     ;EVERY THING IS SET AND READY TO GO
     RET                ;TERMINATE
HINT ENDP
PROG ENDS
     END
```

โปรแกรม 6 โปรแกรมซี สำหรับการรับส่งข้อมูลผ่านทางช่องทางสื่อสารโดยใช้ฮาร์ดแวร์อินเตอร์รัพต์เข้าช่วย

```
/* DEMO PROGRAM HOW TO LINK C TO THE INTERRUPT ROUTINE
   DEFINED IN ASSEMBLY LANGUAGE
   THIS PROGRAM GET INPUT FROM KEY BOARD AND SEND
   OVER THE COMMUNICATION LINE.
   DATA IS RECIEVED VIA INTERRUPT ROUTINE DEFINED IN COMINT.ASM
   THE RECEIVED FLAG SET BY THE INTERRUPT ROUTINE TO TELL THE MAIN
   PROGRAM THAT 1 LINE OF DATA IS RECEIVED.

   COMPILE THIS PROGRAM BY THE MICROSOFT C VERSION 1
   A>MC1 PROG6
   A>MC2 PROG6
   THEN LINK THIS PROGRAM WITH THE ASSEMBLY ROUTINE
   A>LINK C+PROG6+COMINT,PROG6,,MC

   DR.PAISAN SANGUANMU 12-07-87
*/
#include <stdio.h>;
extern char rxbuff[128];   /*this buffer is defined in assembly */
extern int rflag ;         /*rflag is defined in the assembly   */
int     txptr;
char    txbuff[128];

main()

{
int stat,i,c;

 /*set communication buad rate */
 stat =inp(0x3fb);
                                /*set bit #7 to 1 */
 outp(0x3fb,stat | 128);        /*then write to LCR */
 outp(0x3f8,0x0c);              /*set baud rate to 9600*/
 outp(0x3f9,00);
 outp(0x3fb,stat);              /*restore old default parameter*/
 /**set the interrupt vector table***/
 hint();                        /*call the assembly routine*/
 inp(0x3f8);                    /*clear any input that may be there*/
 outp(0x3fc,0x08);              /*let the inerrupt signal get out
                                  from 8250*/
 printf("Communication ready to go\n");
 txptr=0;
 rflag=0;

 while(1)
   {
    if( bdos(11,0)!=0)             /**any key struck? */
        { c=bdos(1,0);             /*get character from key board and
                                     echo on screen*/

          if(c==27)break;          /*if Esc. is hit then exit  */
          txbuff[txptr]=c;         /*collect data in the buffer*/
          txptr++;                 /*increment pointer*/
          if(c==13)send();         /*if carriage return is hit then
                                     send the whole line through
                                     communication port */
        }
    if(rflag==1)                   /*communication received? */
      {
        printf("Received data is:-\n %s",&rxbuff[0]);
        rflag=0;                   /*reset rflag back to 0 */
        }
     }
     outp(0x3fc,0);                /*turn off the interrupt signal gate*/
     exit(0);
   }

/*send string over the communication line*/
send()
 {
  int i;
  for(i=0;i<txptr;i++)             /*loop until end of data*/
    {while( (inp(0x3fd)&32)!=32);  /*loop until 8250 ready to receive new
                                     character */
     outp(0x3f8,txbuff[i]);        /*send character to 8250*/
     }
  txptr=0;                         /*reset pointer*/
  printf("\n");                    /*print new line*/
  return(0);
  }
```